EXP.NO.09: Implement a program to Representation of Intermediate Code Using Three Address Code

```
#include <iostream>
#include <string>
#include <vector>
#include <sstream>

using namespace std;

// Class to represent a Three-Address Code instruction
class TAC {
public:
    string result;  // Result of the instruction
    string operand1; // First operand
    string operand2; // Second operand
    string op;       // Operator

    // Constructor for binary operations
    TAC(string res, string op1, string oper, string op2)
        : result(res), operand1(op1), op(oper), operand2(op2) {}

    // Constructor for assignments (only one operand)
    TAC(string res, string op1)
        : result(res), operand1(op1), op(""), operand2("") {}

    // Function to display the TAC instruction
    void display() {
        if (operand2.empty()) {
            cout << result << " = " << operand1 << endl;
```

```
        } else {
            cout << result << " = " << operand1 << " " << op << " " << operand2 << endl;
        }
    }
};

// Function to generate Three-Address Code for a given expression
void generateTAC(string expression) {
    vector<TAC> tacList;
    int tempVarCount = 1;
    // For this simple example, we manually break down the expression into TAC steps
    // Assuming expression "A = B + C * D"
    string tempVar1 = "t" + to_string(tempVarCount++);
    string tempVar2 = "t" + to_string(tempVarCount++);

    // Step 1: t1 = C * D
    tacList.push_back(TAC(tempVar1, "C", "*", "D"));

    // Step 2: t2 = B + t1
    tacList.push_back(TAC(tempVar2, "B", "+", tempVar1));

    // Step 3: A = t2
    tacList.push_back(TAC("A", tempVar2));

    // Display the TAC instructions
    for (const auto& tac : tacList) {
        tac.display();
    }
```

```
}

int main() {
    string expression = "A = B + C * D";

    cout << "Three Address Code for expression: " << expression << endl;
    generateTAC(expression);
    return 0;
}
```

```
Three Address Code for expression: A = B + C * D
t1 = C * D
t2 = B + t1
A = t2
```